The Java Optimized Processor: Java in a Field-Programmable Gate Array

Schoeberl, Martin

Publication date:
2010

Document Version
Publisher's PDF, also known as Version of record

The Java Optimized Processor: Java in a Field-Programmable Gate Array

Martin Schoeberl
Technical University of Denmark

JavaOne 2010

Overview

- Real-Time Systems and JSR 302
- JOP architecture
- WCET analysis
- Low-level I/O
- Short demo
- Current and future work
- Conclusion

Real-Time Systems

- A definition by John A. Stankovic:

In real-time computing the correctness of the system depends not only on the logical result of the computation but also on the time at which the result is produced.

Real-Time Systems

- Imagine a car accident
  - What happens when the airbag is fired too late?
  - Even one ms too late is too late!
- Timing is an important property
- *Conservative* programming styles

RT System Properties

- Often safety critical
- Execution time has to be known
  - Analyzable system
    - Application software
    - Scheduling
    - Hardware properties
  - Worst-case execution time (WCET)

Issues with COTS

- COTS are for average case performance
  - *Make the common case fast*
  - Very complex to analyze WCET
    - Pipeline (out-of-order)
    - Cache
    - Multiple execution units

The Idea

- Build a processor for RT System
  - *Optimize for the worst case*
- Design philosophy
  - Only WCET analyzable features
    - No unbound pipeline effects
    - New cache structure
  - Shall not be *slow*

Real-Time in Java

- RTSJ (JSR 1)
  - For mixed RT systems
  - Currently updated to version 1.1
- Safety critical Java
  - Target high integrity systems
  - Way smaller, less complex system

Safety Critical Java

- Certification for DO-178B level A
- Java Specification Request 302
  - Lead Dough Locke
- Restricted subset of RTSJ
  - More worst case analysis friendly
  - JSR 302 public draft on the way
- 3 different levels

SCJ Levels

- L0 Single threaded
  - Cyclic executive
- L1 Static threads
  - Initialization and mission phase
  - Ravenscar Ada like
  - No wait/notify
- L2 Multiple missions

SCJ Memory Model

- No Garbage Collection
- RTSJ immortal memory
- RTSJ style scoped memories
  - Scopes are thread private
  - Communication via immortal
- Type system to avoid scope checks

SCJ Execution Model

- Initialization phase - not time critical
  - Class initializing
  - Setup of all data structures and threads
- Mission phase
  - Time critical part
  - Mission can be restarted
  - Level 2: nested missions
    - More dynamic systems

SCJ Tasks

- No threads at level 0
- Static threads/EH, priorities
- Event handlers
  - Time-triggered periodic
  - Event-triggered sporadic
- Single run method for all tasks
  - No waitForNextPeriod()
  - No local state preserving

SCJ Status

- Public draft finished
  - Should get out in the next weeks
- RI almost done
- TCK on the way
- Implementations on the way

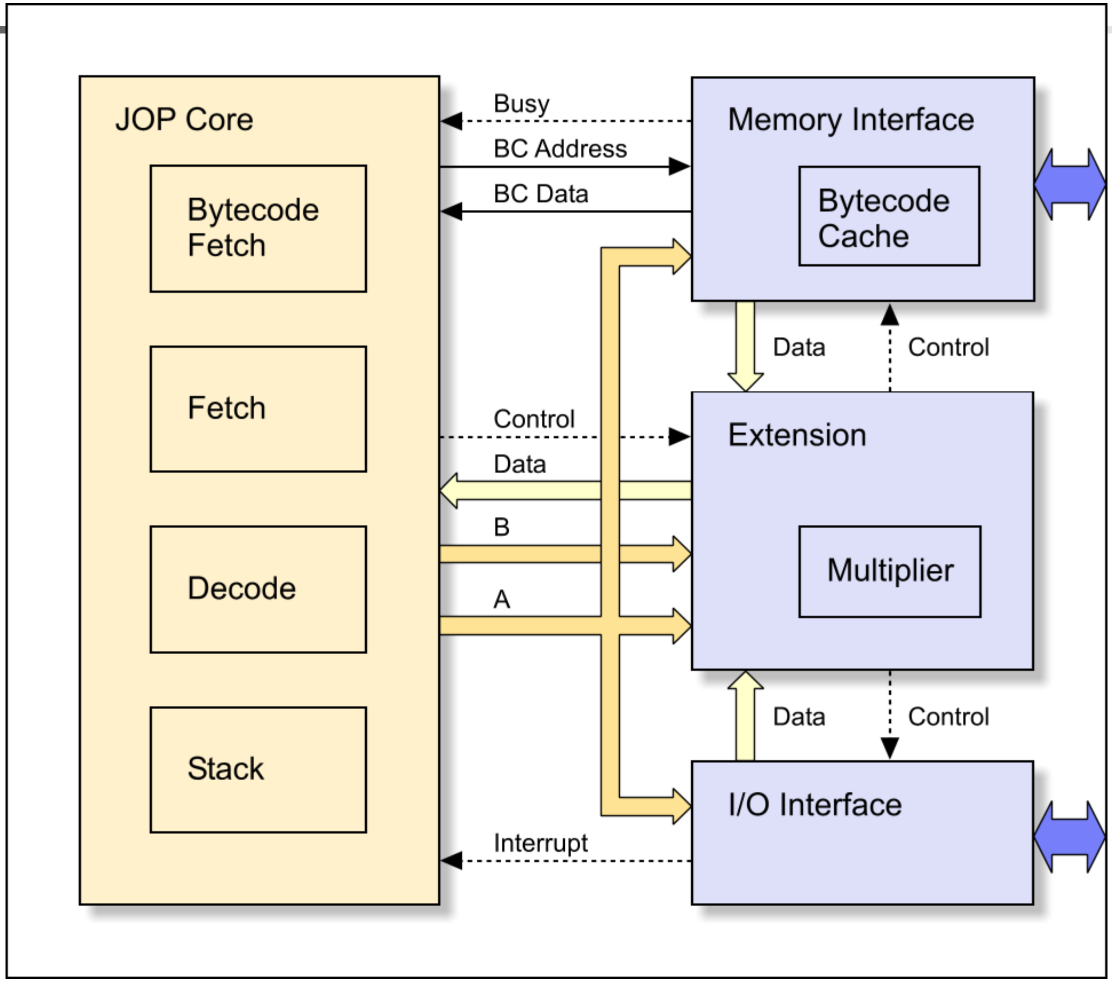
The Java Processor JOP

- For real-time Java
- Support WCET analysis
- Target SCJ (JSR302)
- Implementation in low-cost FPGA

Related Work

- picoJava
  - Sun, no product, released open-source
- aJile JEMCore
  - Available, two versions
- Komodo/jamuth
  - Multithreaded Java processor
- FemtoJava
  - Application specific processor

JOP Block Diagram



JVM Bytecode Issue

- Simple and complex instruction mix
- No bytecodes for *native* functions
- Common solution (e.g. in picoJava):
  - Implement a subset of the bytecodes
  - SW trap on complex instructions
  - Overhead for the trap – 16 to 926 cycles
  - Additional instructions (115!)

JOP Solution

- Translation to microcode in hardware
- Additional pipeline stage
- No overhead for complex bytecodes
  - 1 to 1 mapping results in single cycle execution
  - Microcode sequence for more complex bytecodes
- Bytecodes can be implemented in Java

Microcode

- Stack-oriented
- Compact
- Constant length
- Single cycle
- Low-level HW access

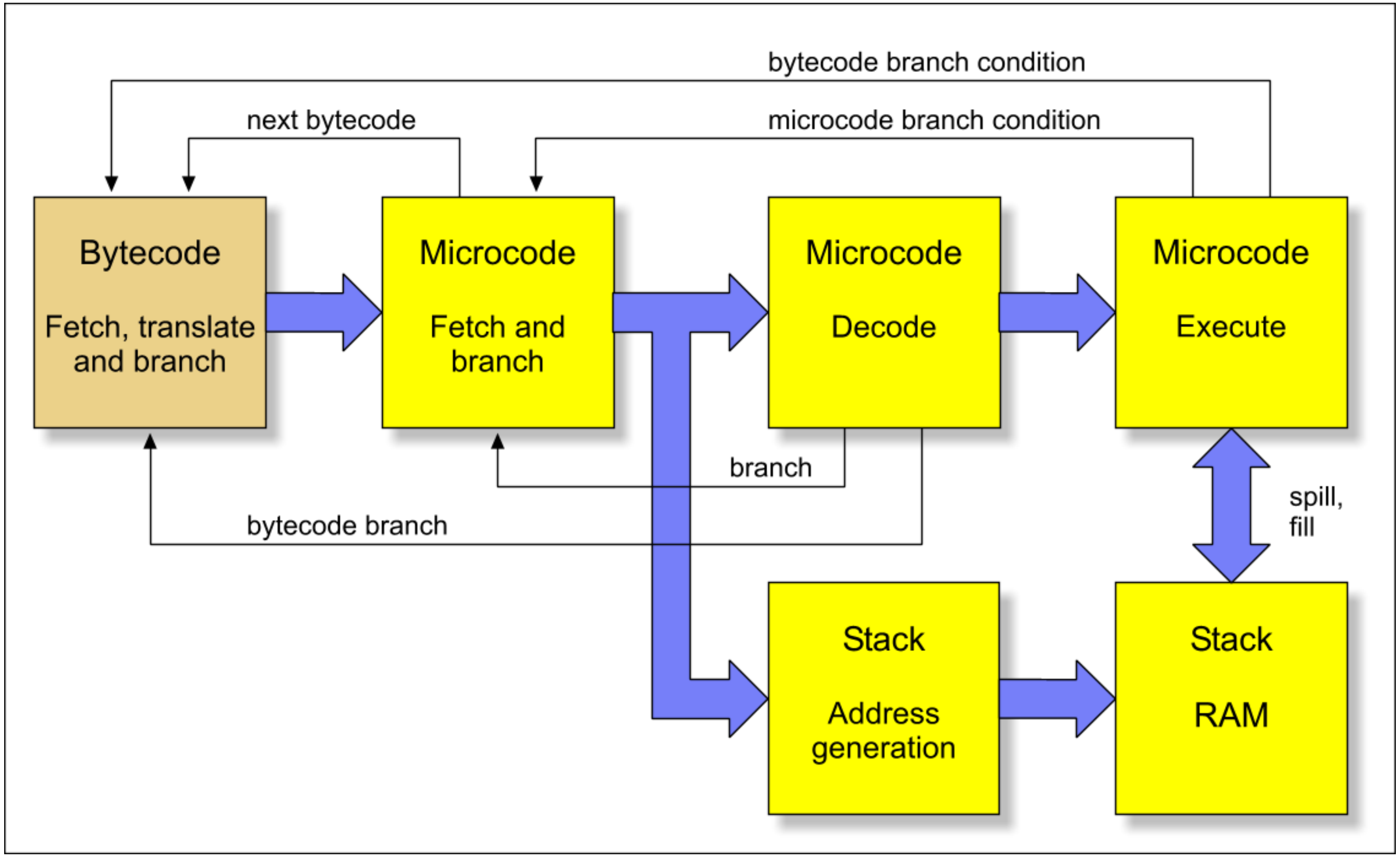
- Two examples

```
dup: dup nxt // 1 to 1 mapping

// a and b are scratch variables
// for the JVM microcode.

dup_x1: stm a     // save TOS
        stm b     // and TOS-1
        ldm a     // duplicate TOS
        ldm b     // restore TOS-1
        ldm a nxt // restore TOS
        // and fetch next bytecode
```

Processor Pipeline



JVM Properties

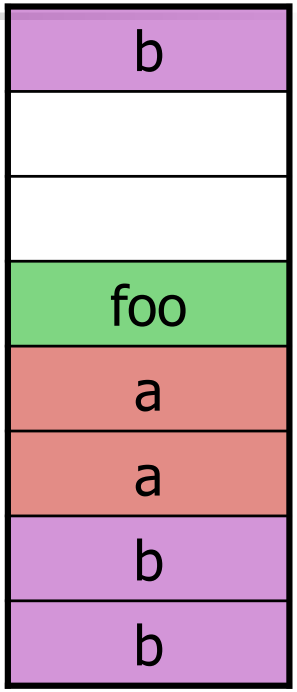
- Short methods
- Maximum method size is restricted
- No branches out of or into a method
- Only relative branches

Proposed Cache Solution

- Full method cached
- Cache fill on call and return
  - Cache misses only at these bytecodes
- Relative addressing
  - Any position in the cache
- No fast tag memory
- Simpler WCET analysis

Method Cache

- Whole method loaded
- Cache is divided in blocks
- Method can span several blocks
- Continuous blocks for a method
- Replacement
  - LRU not useful
  - *Free* running next block counter
  - Stack oriented next block
- Tag memory: One entry per block



Architecture Summary

- Microcode
- 1+3 stage pipeline
- Method cache

The JVM is a CISC stack architecture, whereas JOP is a RISC stack architecture.

WCET Analysis

- WCET has to be known
  - Needed for schedulability analysis
  - Measurement usually not possible
    - Would require test of all possible cases
- Static analysis
  - Theory is mature
  - Low-level analysis is the issue

WCET Analysis

- Path analysis
  - The control flow graph
- Low-level analysis
  - Bytecodes, basic blocks
- Global low-level analysis
  - Cache
- WCET Calculation

WCET Analysis for JOP

- Simple low-level analysis
- Bytecodes are independent
  - No shared state
  - No timing anomalies
- Bytecode timing is known and documented
- Simpler caches

WCET Tool

- Execution time of basic blocks
- Annotated loop bounds (or use DFA)
- Integer linear programming problem solved
- Simple method cache analysis included
  - All methods fit in local scope
    - Single miss
  - Expand local scope

Device Driver in Java

- Solve the chicken-egg problem
  - No device drivers => we need an OS
  - OS forbids low-level access => no device drivers in Java
- Safer than in C – no pointers
- We need:
  - Access to device registers
  - Interrupt handling

Hardware Objects

- Represent a hardware device by a plain Java object
- Device registers are mapped to object fields
- Access with `getfield` and `putfield` bytecode
- Java array for device memory (e.g. video frame buffer)
- Full Java safety
  - Object layout regulates access
  - Bound checks on arrays

An Example – the HWO

```
public final class SerialPort extends HardwareObject {

    public static final int MASK_TDRE = 1;
    public static final int MASK_RDRF = 2;

    public volatile int status;
    public volatile int data;
}
```

An Example - Usage

```
public class Example {

    public static void main(String[] args) {

        IOFactory fact = IOFactory.getFactory();
        SerialPort sp = fact.getSerialPort();

        String hello = "Hello World!";

        for (int i=0; i<hello.length(); ++i) {
            // busy wait on transmit data register empty
            while ((sp.control & SerialPort.MASK_TDRE)==0)
                ;
            // write a character
            sp.data = hello.charAt(i);
        }
    }
}
```

Implementation

- Various JVMs
  - SimpleRTJ – interpreting, no OS JVM
  - JOP – Java processor
  - CACAO – research JIT only JVM
  - OVM – Purdue RTSJ compliant JVM
  - Kaffe – open source JVM
- Effort between a few hours to a few days

Use Case Example

- Serial port
- TCP/IP stack in Java
  - Use SLIP with the serial port
  - Tiny web server
- Runs unchanged on all platforms!
- Full Java solution

Applications

- Tilt of railway power line
  - Distributed motor control

- Austrian Railway
  - Train control system
  - GPS, GPRS, supervision
- TeleAlarm
  - Remote tele-control
  - Data logging
  - Automation

Demo

- Time for a short JOP/WCET Demo

Current/Future Work

- JOP CMP
  - 12 cores in a low-cost FPGA runnig
  - Time-triggered memory access
- Analyzable D$
  - Pressure due to CMP
  - Heap access hard to analyze
- Hardware transactional memory
  - Static analyzable retry limit

Conclusions

- Real-time Java processor
  - Exactly known execution time of the BCs
  - Time-predictable method cache
  - WCET analysis possible
- Resource-constrained processor
  - RISC stack architecture
- In industrial use
- Platform for RT architecture research

More Information

- JOP Thesis and source
  - http://www.jopdesign.com/thesis/index.jsp
  - http://www.jopdesign.com/download.jsp
- Various papers
  - http://www.jopdesign.com/docu.jsp
- Web Sites
  - http://www.jopdesign.com/
  - http://www.jopwiki.com/

Thank You!

Questions?